Three Headed Mastery: minBERT as a Jack of All Trades in Multi-Task NLP

Stanford CS224N Default Project

Valerie Fanelle[1], **Rafael Perez Martinez**[2], **and Ifdita Hasan Orney**[3]
Graduate School of Business[1], Dept. of Electrical Engineering[2], Dept. of Computer Science[3]
Stanford University
{vfanelle, rafapm, ifdi1101}@stanford.edu

Mentor: Nelson Liu No External Collaborators No shared project

Abstract

The Bidirectional Encoder Representations from Transformers (BERT) model, introduced by Devlin et al. (2019), is a transformer-based model renowned for its remarkable performance across a variety of natural language processing (NLP) tasks, including sentiment analysis, paraphrase detection, and Semantic Textual Similarity. This groundbreaking model set new benchmarks in NLP upon its release in 2018. Building on this foundational work, we delve into minBERT, a "smaller" variant of the original BERT model. Our objective is to experiment with various approaches to enhance the performance of the minBERT model without being hindered by expensive computational demands. We combine various approaches that include parallel training, more sophisticated optimizers (e.g., PyTorch's AdamW and Google Brain's Lion), and advanced architectures to enhance accuracy and efficiency in sentence-level tasks. This strategy also involves fine-tuning critical hyperparameters and implementing methods like dropout and early stopping to prevent overfitting and ensure robust generalization. Lastly, we evaluate our minBERT model using three datasets: Stanford Sentiment Treebank, Quora Question Pairs, and Semantic Text Similarity. Our best-performing model obtains an overall score of 0.731 and 0.734 in the dev and test leaderboards, respectively.

1 Introduction

Bidirectional Encoder Representations from Transformers (BERT) by Devlin et al. (2019) is a transformer-based model that has performed remarkedly well in various natural language processing (NLP) tasks such as sentiment analysis, paraphrase detection, and Semantic Textual Similarity. This model advanced the state-of-the-art (SOTA) in the field when it was released in 2018. In the original BERT paper by Devlin et al. (2019), two model sizes were presented: $\text{BERT}_{\text{base}}$ and $\text{BERT}_{\text{large}}$, containing 110 million and 340 million parameters, respectively. We use minBERT, a simplified and minimalist version of the original BERT framework. This approach enables us to identify the bottlenecks inherent in the original BERT model's architecture, allowing for a more focused exploration of potential improvements without the limitations imposed by extensive computational needs that a larger model would require.

This paper summarizes a series of extensions to a baseline minBERT model to optimize its performance across multiple sentence-level tasks, more specifically, sentiment analysis, paraphrase detection, and semantic textual similarity. We began by implementing multi-task training with sequential training. However, we faced challenges with catastrophic forgetting during training since one of the datasets we considered was disproportionately large compared to the other two. To partially mitigate this issue, we shifted to parallel training, i.e., where smaller datasets are iterated multiple times within one epoch of the most extensive dataset to ensure that an equal number of batches are produced across all three datasets.

We further refined our model's architecture by adopting more advanced optimizers such as PyTorch's (Foundation, 2024) implementation of Adam (Kingma and Ba, 2017) with decoupled Weight decay (Loshchilov and Hutter, 2017), which is commonly referred to as AdamW, and Google Brain's Lion (Evo**L**ved S**i**gn M**o**me**n**tum, Chen et al. (2023)). These optimizers showcased better training efficiency and model convergence, which led to an overall score improvement. Once we achieved satisfactory preliminary results above the baseline model, we considered various multi-headed model architectures, which we implemented along with our parallel training approach and advanced optimizers. This allowed us to obtain our highest overall dev and test scores of 0.731 and 0.734, respectively.

2 Related Work

After the introduction of the BERT model, there has been a resurgence of more advanced BERT model variants such as RoBERTa (Liu et al., 2019), ALBERT (Lan et al., 2020), and ELECTRA (Clark et al., 2020), which target specific NLP tasks through a variety of multi-task learning (MTL) approaches. Some of these methods include simple additional neural network layers on top of the pre-trained model as well as adding more complex and sophisticated architectural designs to the model to excel at tasks such as sentiment analysis, paraphrase detection, and Semantic Textual Similarity. For example, we took inspiration from other advanced BERT variants, such as Sentence-BERT (Reimers and Gurevych, 2019) and TwinBERT (Lu et al., 2020), which introduced more specialized architectures targeting tasks such as semantic textual similarity by utilizing sophisticated approaches like embedding comparisons using cosine similarity and mean pooling strategies.

We initially faced challenges due to uneven dataset sizes for the different tasks we considered. Previous works have addressed this issue by using a variety of techniques to overcome different dataset sizes during pre-training and fine-tuning. As such, we explored a parallel training approach, which was inspired by the work of Stickland and Murray (2019), where they modified the multi-task training routine to sample the smaller datasets more times during one epoch.

In addition to these changes to the model's architecture, we considered other more advanced optimizers besides the one we implemented from scratch during the milestone (AdamW) (Kingma and Ba, 2017). We first considered PyTorch's version of AdamW, which resulted in a better performance. However, in recent years, progress has been made toward developing optimizers that outperform Adam, especially in the context of language modeling and computer vision. Some of these optimizers include Google Brain's Lion optimizer and Stanford's Sophia (Scalable Stochastic Second-order Optimizer for Language Model Pre-training, Liu et al. (2024)). We primarily investigated Lion, as previous works with this optimizer have yielded promising results in various ML tasks, including language modeling (Chen et al., 2023).

3 Approach

3.1 Baseline model

We begin by implementing a baseline version of minBERT to benchmark our extension's improvements. Below, we describe in detail each of the changes we implemented for the three classification tasks. For sentiment analysis, the goal is to predict the sentiment of given sentences across five classes (from negative to positive). We start by computing the sentence embedding by passing the respective inputs to a *forward* function. Specifically, the *forward* function processes batches of sentences using the BERT model, taking input_ids and attention_mask as inputs. It then extracts the [CLS] token's embeddings, which represent the entire sentence. These embeddings are then used as foundational representations for the various task-specific analyses in sentiment analysis and the other two tasks. After generating the sentence embeddings, we then apply a linear layer which processes the embeddings to produce five logits per sentence, representing the model's confidence in each of the sentiment classes. This is then followed by a dropout layer which we apply to these embeddings to mitigate overfitting. Lastly, we used the logits to classify the sentiment of each sentence.

In the context of paraphrase detection, we are comparing a pair of sentences to see if they have the same meaning. We start by computing the embeddings of each sentence using the *forward* function. We then calculate the absolute difference between the embeddings of the pair of sentences to quantify if there are any discrepancies. A linear layer is applied to produce a logit that indicates the likelihood

of the sentences having similar meanings (i.e., they are paraphrases), followed by dropout to prevent overfitting. For the semantic textual similarity task, our goal is to assess the similarity between pairs of sentences. We do this by calculating the embeddings for each sentence in the pair and then finding the absolute difference between these embeddings (a similar approach taken in paraphrase detection). We apply a linear layer, which processes these embeddings to ultimately produce a logit that represents the similarity between the sentences, followed by dropout.

Once the baseline minBERT model has been established, we initiate an initial training procedure, which will be refined and modified in subsequent extensions. In our initial approach, we implemented from scratch the AdamW optimizer (Kingma and Ba, 2017). This optimizer particularly enhances the model's training efficiency by dynamically adapting the learning rates, incorporating momentum to dampen oscillations, and accelerating the optimizer in the desired direction.

Since our goal is to enhance the minBERT model's performance across three downstream tasks, we first employ a basic multi-task learning framework. In this particular framework, we simultaneously consider the loss of all three tasks, i.e., the total loss is calculated $\mathcal{L}$ by aggregating the losses for each task. This is represented mathematically as:

$$\mathcal{L} = \mathcal{L}_{\text{SA}} + \mathcal{L}_{\text{PD}} + \mathcal{L}_{\text{STS}}, \tag{1}$$

where L_{SA} is the cross-entropy loss, L_{PD} is the binary cross-entropy loss, and L_{STS} is the mean-squared error loss for the Sentiment Analysis (SA), Paraphrase Detection (PD), and Semantic textual similarity (STS) tasks, respectively. We particularly considered a "sequential" training approach where we shift between tasks during each epoch. This is showcased in the diagram of Fig. 1(a).

3.2 Parallel Multi-Task Training Routine

Given the difference in dataset sizes for each different task, we initially ran into catastrophic forgetting, which is defined in Luo et al. (2023) as "a phenomenon that occurs in machine learning when a model forgets previously learned information as it learns new information." This was particularly the case in our baseline model since the dataset used to fine-tune the paraphrase detection task was relatively larger than the dataset of the other two tasks. As such, we implemented a parallel multi-task training routine. In this routine, we iterated multiple times through the smaller datasets within one epoch of the most extensive dataset to ensure that an equal number of batches were produced across all three datasets. This is showcased in Fig. 1(b), along with the sequential training routine.

3.3 Task Specific Heads

While our baseline model considered three simple task-specific heads, the performance for the three tasks was unsatisfactory. As such, we implemented several iterations of task-specific heads to improve the performance until we reached an overall score higher than 0.7 in the dev and test leaderboards.

For the sentiment analysis task, we started with a basic architecture, leveraging embeddings produced by BERT. Our initial approach involved integrating a 1D convolutional layer after the BERT embeddings, anticipating that the convolutions would capture local context that is helpful for sentiment analysis. The Conv1D layer was meant to process the sequence of the embeddings outputted by BERT, hoping to enhance feature extraction specific to the sentiment task. We chose this architecture because we believed a convolutional approach could add value by emphasizing important n-gram features within the sentences (Saba and Rosales, 2023). However, the convolutional model did not deliver the improvements we were seeking. So, we pivoted to a simpler architecture inspired by SBERT, which is known for its effectiveness in semantic textual similarity tasks. SBERT's approach to creating sentence embeddings through mean pooling of BERT's output provided a less complex and more computationally efficient pathway to sentence representation. We used this strategy within the forward function to form sentence embeddings for all task-specific heads, which included the sentiment analysis head. Forgoing the convolutional layers in favor of simplicity and mean pooling improved our model's performance. With this change, we achieved a balance between model complexity and computational efficiency, ultimately surpassing the 0.7 threshold on the development and test leaderboards.

Paraphrase detection had the challenge of understanding the semantic equivalence between sentences. Our baseline model passed BERT's embeddings through a simple linear layer, but this approach was not enough to consistently identify paraphrased text pairs. To enhance our paraphrase detection head,

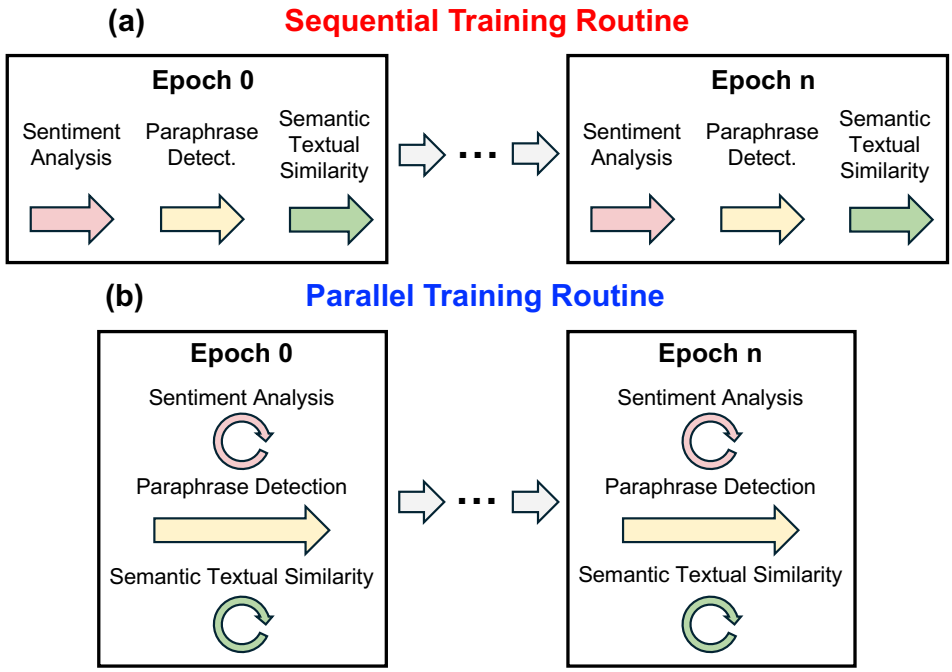

Figure 1: (a) Sequential and (b) parallel training routines for the minBERT model.

our model followed a layered approach, beginning with BERT to process sentence pairs, followed by an advanced head structure for paraphrase detection. These embeddings were passed through a head consisting of the following layers:

1. **Linear Transformation**: A dense layer reduced the high-dimensional BERT embeddings to a smaller feature set. This step aimed to condense the essential information for paraphrase detection into a more compact representation.
2. **Activation Function**: A ReLU activation added non-linearity to the model, helping to capture complex patterns in the data.
3. **Dropout**: To combat overfitting, a dropout layer was included, which randomly set a portion of the input units to zero during training, forcing the model to learn more robust features.
4. **Final Linear Layer**: Another dense layer transformed the features to the final logit.
5. **Sigmoid Activation**: Convert the output to a probability between 0 and 1, given this is a binary classification task.

This initial paraphrase detection head did not perform as well as anticipated. Despite its theoretical robustness, the added layers and parameters may have led to the model overfitting the training data or being too rigid to generalize well to unseen data. Therefore, we pivoted to a more simplified approach. We calculated the absolute difference between BERT's sentence embeddings to emphasize the discrepancies between sentence pairs. This strategy aimed to directly capture the degree of variation between the two sentences, working under the assumption that true paraphrases would exhibit minimal differences in their BERT representations.

The task-specific head was thus reduced to a single intermediate linear layer that mapped the BERT embeddings to a lower-dimensional space conducive to binary classification. This was followed by a GELU activation function, known for its smooth non-linearity and efficient training performance. The output of this activation function was then fed into a final linear layer that compacted the features down to a single logit representing the probability of the sentence pair being a paraphrase. The design was completed with a sigmoid activation applied to the logit, outputting a probability score between 0 and 1. By concentrating on the essential components of the paraphrase detection task—namely, the relative difference between embeddings and their transformation into a probability score—this final architecture demonstrated improved performance metrics and generalization capabilities on our development and test datasets.

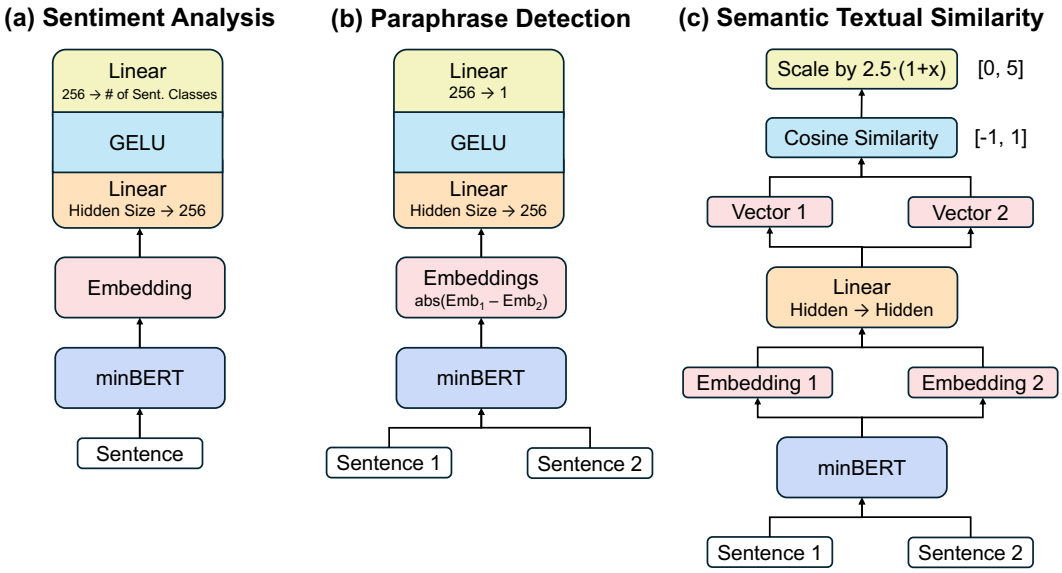

Figure 2: Model architecture of the proposed minBERT model for (a) semantic analysis, (b) paraphrase detection, and (c) semantic textual similarity tasks.

For the semantic text similarity task, we processed the sentence pairs through BERT to get their embeddings and then passed them into a two-step linear transformation process, punctuated with ReLU activation and dropout for non-linearity and regularization. The linear layers were intended to compress and then expand the embeddings, hypothesizing that this would allow the model to reconstruct and emphasize features important in semantic similarity:

1. A first linear layer to compress the embeddings from a higher to a lower dimension.
2. A ReLU activation function to introduce non-linearity.
3. A dropout layer to prevent overfitting.
4. A second linear layer to map the features back to the original embedding dimension.

After the transformation, we applied a sigmoid function and scaled the result to the range of [0, 5], aiming to map the cosine similarities to the labeled similarity scores as per the STS task requirements.

However, this initial approach did not yield the results we were aiming for. The transformation steps seemed to distort rather than clarify the semantic signals required for accurate similarity scoring. The performance metrics indicated that the complexity of this architecture might have led to overfitting, and the model failed to generalize well on unseen data.

Thus, we shifted our strategy toward a more effective technique inspired by SBERT (Sentence-BERT), which leveraged the mean pooled embeddings from BERT followed by a cosine similarity computation for the STS task. By adopting mean pooling, we could maintain the richness of BERT's contextual embeddings while still capturing the essence of sentence similarity. The cosine similarity provided a straightforward yet powerful measure that correlated well with human judgment on the task, as demonstrated by SBERT's success.

Our final implementation demonstrated significant improvement in our STS performance. It showed that a well-chosen, straightforward method could outperform more complex ones, affirming the importance of model selection based on task-specific characteristics and empirical results. Following this, the final implementation of our model, illustrated in Figure 2, represents the culmination of an iterative development and empirical validation process. This process underscores its robustness, achieving the highest accuracy on both development and test leaderboards.

3.4 Advanced Optimizers

Optimizers are fundamental for training neural networks, particularly language models like BERT. Over the past decade, several optimization algorithms have emerged, but only two have become the

gold standards among many others: AdamW and Adafactor. Both of these optimizers are frequently used by ML practitioners across various domains, ranging from language to vision to multimodal systems. In our baseline model, we have implemented the AdamW optimizer from scratch. However, a more sophisticated implementation of AdamW is available within PyTorch (Foundation, 2024).

As part of one of our extensions, we shifted towards using PyTorch's version of the AdamW optimizer once we had achieved satisfactory preliminary results beyond the baseline model. We particularly experimented with a combination of different optimizers along with various extensions.

In addition, we also tested Lion, an optimizer proposed by the Google Brain team (Chen et al., 2023). Unlike AdamW, this optimizer stands out for its memory efficiency as it only keeps track of the *momentum*. In particular, Lion updates each parameter with a uniform magnitude determined through sign operation. One of the most significant advantages of this optimizer is that it can achieve similar performance to AdamW with a 2x speedup (i.e., a faster runtime). It is also more straightforward to compare AdamW to Lion as they share almost identical hyperparameters, with the exception of ϵ, which is not included in Lion. The primary difference between these two optimizers is that the learning rate for Lion is usually 3 to 10 times smaller than that for AdamW, which also impacts the weight decay (i.e., it will be 3 to 10 times larger to maintain the same $\alpha \cdot \lambda$ product). In addition, the default β_2 factor is 0.99 instead of 0.999 (Adam's default β_2 value).

We also implemented Surgical Fine-tuning (Lee et al., 2023), a specialized optimization technique. We froze the first few layers of the pre-trained model from experiment 7 and fine-tuned it using the three datasets along with batch size 8 to reduce overfitting.

4 Experiments

4.1 Data and Evaluation Metrics

Experiments were conducted on three datasets: the Stanford Sentiment Treebank (Socher et al., 2013), binary paraphrase detection on Quora Question Pairs (QQP, Quora (2017)), and regression on Semantic Text Similarity (STS, Agirre et al. (2013)). Details and evaluation metrics of these datasets are described below:

Stanford Sentiment Treebank: Used for sentiment analysis by classifying movie reviews into five different categories: negative, somewhat negative, neutral, somewhat positive, or positive. It was parsed using the Stanford parser and was annotated by three human judges. It consists of 11,855 single sentences from movie reviews, and the splits include training (8,544 examples), development (1,101 examples), and test (2,210 examples) sets. Here, accuracy is used as the evaluation metric (Socher et al., 2013).

Quora Question Pairs: Used for paraphrase detection to identify whether question pairs are paraphrases of one another. It contains a total of 400,000 question pairs. The dataset is divided into training (141,506 examples), development (20,215 examples), and test (40,431 examples) sets. Model performance is evaluated by accuracy (Quora, 2017).

Semantic Text Similarity: Used to measure sentence pair similarity on a 0 to 5 scale. A total of 8,628 different sentence pairs were split into training (6,041 examples), development (864 examples), and test (1,726 examples) sets. The Pearson correlation coefficient is used as the evaluation metric (Agirre et al., 2013).

4.2 Experimental Details

For our experiments, we shared some common settings across the board. For our training procedures, we ran ten epochs utilizing either pre-training or fine-tuning procedures. A learning rate (α) of 1e-3 was chosen for pre-training, while an $\alpha =$ 1e-5 was selected for fine-tuning unless stated otherwise. We also used a hidden-layer dropout rate of 0.3. For batch size, we considered various sizes, but for the most part, we used either a batch size of 40 or 64 for pre-training and a batch size of 8 or 35 for fine-tuning (depending on memory constraints). These values are detailed in the results subsection (4.3)). We trained our models in either the Nvidia T4, V100, or A100 GPUs (depending on their availability in Google Colab).

For the optimizer settings, we used the following hyperparameters for AdamW: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon =$ 1e-6, and $\lambda = 0$. For Lion, we used slightly different values: α = 1e-4 (for pre-training) or 1e-6 (for fine-tuning), $\beta_1 = 0.9$, $\beta_2 = 0.99$, and $\lambda = 0$. The recommended values for the learning rates of the Lion optimizer were 3-10x lower. We tried both extremes and noticed that we obtained superior performance by using a 10x lower value. We also tried testing a variety of hyperparameter values for hidden-layer dropout rate, learning rate, λ, but we noticed that either we obtained a similar score or it got worse, so we decided to stick to these values due to limited computational resources.

4.3 Results

Table 1: Detailed Performance Metrics by Experiment on Development Dataset

#	Experiment	Overall Accuracy	SA	PD	STS	Batch Size
Baseline Heads and Sequential Training						
1	Sequential (pre-trained)	0.465	0.292	0.624	-0.043	40
2	Sequential (fine-tuned)	0.534	0.459	0.625	0.037	8
Baseline Heads and Parallel Training						
3	Parallel training (pre-trained)	0.548	0.351	0.625	0.337	40
4	Parallel training (fine-tuned)	0.602	0.500	0.625	0.363	8
Task-Specific Heads, Parallel Training, and No Optimizer						
5	Exp. 4 + Three Heads Are Better Than One Paper Architecture	0.478	0.190	0.625	0.236	40
6	Exp. 4 + Sentiment-and-Paraphrase Custom Heads	0.684	0.515	0.841	0.393	40
Additional Task-Specific Heads, Parallel Training, and Optimizer						
7	Exp. 6 + STS Custom Head + AdamW	**0.731**	0.502	**0.865**	0.643	8
8	Exp. 7 + Higher α (2e-5)	0.729	0.491	0.858	0.672	8
9	Exp. 7 + Additional STS Layers	0.668	0.488	0.636	**0.761**	35
Advanced Optimizers						
10	Exp. 6 + PyTorch's AdamW	0.699	**0.526**	0.841	0.457	8
11	Exp. 6 + Lion	0.643	0.477	0.778	0.350	64
12	Exp. 9 + Lion	0.658	0.488	0.722	0.525	64
13	Exp. 7 + Weighted loss function	0.714	0.513	0.862	0.530	8
14	Exp. 7 + Surgical Fine-tuning	0.725	0.481	0.862	0.660	8

Table 2: Results from Highest-Performing Model Architecture on Test Dataset

Overall Accuracy	SA	PD	STS	Batch Size	Learning Rate	Optimizer
0.734	0.520	0.865	0.637	8	1e-5	PyTorch's AdamW

We conducted the above 14 experiments and obtained these results, with Experiment #7 being the highest-performing. In the following section, we will analyze these results in detail.

5 Analysis

Our results paint a story about the many levers one can pull to optimize NLP models, such as redesigning the model's architecture to support task-specific heads, various training strategies, and optimization techniques.

Initially, our baseline models with sequential training performed poorly, demonstrating a need for a more robust training routine. The introduction of parallel training marked a significant improvement in overall accuracy, particularly benefitting the Semantic Text Similarity (STS) task, showcasing parallel training's ability to address catastrophic forgetting effectively.

Task-specific modifications to our task heads further propelled our models' capabilities. For paraphrase detection, the transition to calculating absolute differences between sentence embeddings resulted in superior accuracy over the baseline approach. For Semantic Text Similarity, using simple

strategies rooted in NLP theory, such as mean pooling and calculating cosine similarity between sentence pairs, was most effective. For sentiment, using simple layers instead of complex convolutional layers allowed for more robust sentiment detection capabilities as the model became better attuned to the global sentiment representations of the text.

Optimization techniques also played an important role. The switch to AdamW, particularly PyTorch's implementation, marked a noticeable improvement in our overall accuracy score, which could be attributed to better handling of weight decay or more nuanced learning rate adjustments within the optimizer's algorithm. The implementation of surgical fine-tuning further enhanced our model's performance by reducing overfitting, allowing for more generalized and robust results. While effective in reducing overfitting for STS, it resulted in suboptimal performance for the other two tasks. We believe the selective updating of the model layers and parameters might have inadvertently constrained the model's ability to generalize for the other two tasks. The Lion optimizer, while not leading to any top scores, allowed for a balance of performance and efficiency which was beneficial in testing various heads, particularly at larger batch sizes. Notably, the addition of task-specific layers and a deliberate combination of optimization methods led to our highest accuracy scores. This iterative process, marked by empirical testing and methodical adjustments, was critical in identifying the most effective models.

However, our journey was not without its challenges. Complex task-specific heads did not deliver the expected benefits, the weighted loss function did not enhance performance as hypothesized, and the Lion optimizer, despite its promise of efficiency, fell short in comparison to AdamW. Our team switched from the custom project due to a lack of computing resources to run LLaMA 2 and OpenAI's Whisper. As a result, we had little time to apply smart regularization techniques and leverage tools like Optuna for hyperparameter optimization. Again, limited computing resources on the default project also limited our ability to test and experiment quickly with additional optimizations, such as supervised contrastive learning and a prototypical neural network.

In conclusion, the above learnings from our experiments highlight a clear approach to optimizing NLP models: a balanced blend of simplicity in model architecture, task-specific design, strategic training approaches, and judicious selection of optimizers. This journey exemplifies the nuanced interplay between various parts of model development, ultimately leading to a suite of models that excel in their respective NLP tasks while maintaining a synergy that boosts overall performance.

6 Conclusion

Through our exploration of performance enhancements to minBERT, we discovered the great impact model architecture, training routines, and optimizer choice could have on sentence-level NLP tasks. We found that parallel training significantly mitigated catastrophic forgetting issues present in sequential training, resulting in a boost in model accuracy, particularly for Semantic Textual Similarity. Simplified task-specific heads tailored to each NLP task—absolute difference measurements for paraphrase detection and SBERT-inspired techniques for STS—proved superior to our more complex initial approaches. However, we faced limitations, including underutilization of smart regularization techniques and hyper-parameter optimization due to computing constraints, which restricted our ability to iterate rapidly.

In reflection, our project's achievements highlight the delicate balance required between model complexity and task specificity. Future work could benefit from exploring utilizing ensembles to boost task-specific performance, advanced regularization techniques, diversifying training data, and incorporating cutting-edge optimizers to push the boundaries of model efficiency and accuracy. Despite the challenges, our work provides a roadmap for improving NLP models and underscores the iterative nature of machine learning research.

7 Team Contributions

Valerie: Implemented Baseline model, Task-specific heads, and custom heads and ran experiments
Rafael: Implemented Baseline model, fine-tuned using advanced optimizers like PyTorch AdamW, Lion and ran experiments
Ifdita: Implemented the Baseline model, Surgical fine-tuning and ran experiments.

References

Eneko Agirre, Daniel Cer, Mona Diab, Aitor Gonzalez-Agirre, and Weiwei Guo. 2013. *SEM 2013 shared task: Semantic Textual Similarity. In *Second Joint Conference on Lexical and Computational Semantics (*SEM), Volume 1: Proceedings of the Main Conference and the Shared Task: Semantic Textual Similarity*, pages 32–43, Atlanta, Georgia, USA. Association for Computational Linguistics.

Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Yao Liu, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, and Quoc V. Le. 2023. Symbolic Discovery of Optimization Algorithms.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.

The PyTorch Foundation. 2024. torch.optim.AdamW - PyTorch documentation. Accessed: 2024-03-14.

Diederik P. Kingma and Jimmy Ba. 2017. Adam: A Method for Stochastic Optimization.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations.

Yoonho Lee, Annie S. Chen, Fahim Tajwar, Ananya Kumar, Huaxiu Yao, Percy Liang, and Chelsea Finn. 2023. Surgical fine-tuning improves adaptation to distribution shifts.

Hong Liu, Zhiyuan Li, David Hall, Percy Liang, and Tengyu Ma. 2024. Sophia: A Scalable Stochastic Second-order Optimizer for Language Model Pre-training.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach.

Ilya Loshchilov and Frank Hutter. 2017. Decoupled Weight Decay Regularization. *arXiv preprint arXiv:1711.05101*.

Wenhao Lu, Jian Jiao, and Ruofei Zhang. 2020. TwinBERT: Distilling Knowledge to Twin-Structured BERT Models for Efficient Retrieval.

Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang. 2023. An Empirical Study of Catastrophic Forgetting in Large Language Models During Continual Fine-tuning.

Quora. 2017. First quora dataset release: Question pairs. `https://data.quora.com/First-Quora-Dataset-Release-Question-Pairs`. Accessed: 02-28-2024.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks.

Esteban Cambronero Saba and Jesus Meza Rosales. 2023. Three Heads Are Better Than One. Stanford CS224N Default Project.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

Asa Cooper Stickland and Iain Murray. 2019. BERT and PALs: Projected Attention Layers for Efficient Adaptation in Multi-Task Learning.